Cassandra1.0.x
实用教程

2011-12-09

李克喜

适合 

目录

1.数据分析建模

1.0. 例子：产品

字段名称	数据类型（附录）	中文意义	允许为空	是否索引
Id_Product	LongType	产品 Id	Not NULL	KEYS
Product_Name	UTF8Type	产品名称	Not NULL	
Id_Factory	LongType	生产厂家 Id	Not NULL	KEYS
Factory_Name	UTF8Type	生产厂家名称	Not NULL	
Product_Type	UTF8Type	产品类型	Not NULL	KEYS
Product_Place	UTF8Type	产品原产地	NULL	
Product_Price	UTF8Type	产品价格	Not NULL	KEYS
Product_Count	LongType	产品数量	Not NULL	
Product_Income	UTF8Type	进货日期	NULL	KEYS
Product_Batch	UTF8Type	进货批次	NULL	

1.1.例子：商户

字段名称	数据类型（附录）	中文意义	允许为空	是否索引
Id_Factory	LongType	商户 Id	Not NULL	KEYS
Factory_Name	UTF8Type	商户名称	Not NULL	
Factory_Address	UTF8Type	商户地址	Not NULL	
Factory_Tel	UTF8Type	商户电话	Not NULL	
Factory_Type	UTF8Type	商户类型	Not NULL	KEYS
Factory_People	UTF8Type	商户联系人	Not NULL	
Factory_Cooperation	UTF8Type	合作级别	Not NULL	KEYS
Factory_City	UTF8Type	商户所在城市	Not NULL	KEYS

2.Cassandra建立数据库模型

2.0. cassandra-cli客户端操作

1)、建立 KeySpace：CassandraKS<名称可以按字符串规格自由定义，这里暂时去该名字>

注：每次操作尽可能的给出图例以便说明实际操作结果

[default@unknown] Create keyspace CassandraKS;

```
[default@unknown] Create keyspace CassandraKS;
86ee48c0-2240-11e1-0000-242d50cf1fbd
Waiting for schema agreement...
... schemas agree across the cluster
[default@unknown]
```

2）、连接到刚刚创建的 KeySpace。

[default@unknown] use CassandraKS;

```
[default@unknown] use CassandraKS;
Authenticated to keyspace: CassandraKS
[default@CassandraKS]
```

3）、建立列簇（Column family）：Products<对应数据建模的产品表>

[default@CassandraKS] create Column family Products with comparator=UTF8Type
and default_validation_class=UTF8Type and key_validation_class=UTF8Type
and column_metadata=
[
{column_name: Id_Product, validation_class: LongType,index_type: KEYS},
{column_name: Product_Name, validation_class: UTF8Type},
{column_name: Id_Factory, validation_class: LongType,index_type: KEYS},
{column_name: Factory_Name, validation_class: UTF8Type},
{column_name: Product_Type, validation_class: UTF8Type,index_type: KEYS},
{column_name: Product_Place, validation_class: UTF8Type},
{column_name: Product_Price, validation_class: UTF8Type, index_type: KEYS},
{column_name: Product_Count, validation_class: LongType},
{column_name: Product_Income, validation_class: UTF8Type, index_type: KEYS},
{column_name: Product_Batch, validation_class: UTF8Type},
{column_name: rflag, validation_class: UTF8Type, index_type: KEYS}
];

注：comparator – 比较器，用来定义的数据类型，用于验证和排序的列名，这里定义为 UTF-8 字符串型。
default_validation_class – 默认列的数据校验器，定义用来验证列的数据类型，默认为 UTF-8 字符串型，也就是如果在添加列时没有指定列的数据类型，那么列的数据类型默认是 UTF-8 字符串型。
key_validation_class – 默认键验证器，定义的数据类型用来验证 Key 的值，这里定义为 UTF-8 字符串型。
column_metadata – 列的构造属性。
每个列的基本结构是：
{column_name: Id_Product, validation_class: LongType,index_type: KEYS}
符合 JSON 数据格式：Key/Value，这也是 Cassandra 的最大特点。

column_name – 列名的 Key，Id_Product –列名的 Value。
validation_class – 列验证器的 Key，LongType – 列验证器的 Value
index_type – 列索引的Key，KEYS列索引的Value（参考索引分析）

```
[default@CassandraKS] create column family Products with comparator=UTF8Type
...     and default_validation_class=UTF8Type and key_validation_class=UTF8Type
...     and column_metadata=[
...     {column_name: Id_Product, validation_class: LongType,index_type: KEYS},
...     {column_name: Product_Name, validation_class: UTF8Type},
...     {column_name: Id_Factory, validation_class: LongType,index_type: KEYS},
...     {column_name: Factory_Name, validation_class: UTF8Type},
...     {column_name: Product_Type, validation_class: UTF8Type,index_type: KEYS},
...     {column_name: Product_Place, validation_class: UTF8Type},
...     {column_name: Product_Price, validation_class: UTF8Type, index_type: KEYS},
...     {column_name: Product_Count, validation_class: LongType},
...     {column_name: Product_Income, validation_class: UTF8Type, index_type: KEYS},
...     {column_name: Product_Batch, validation_class: UTF8Type}
...     ];
92410240-2244-11e1-0000-242d50cf1fbd
Waiting for schema agreement...
... schemas agree across the cluster
[default@CassandraKS]
```

注：图中“[redacted]”是指换行前的空白，是 Cassandra-cli 自动加上去的，因为 Cassandra-cli 执行命令的结尾是“;”，只有碰到分号 Cassandra-cli 才会执行语句。

4）、建立列簇（Column family）：Factorys<对应数据建模的商户表>

```
[default@CassandraKS] create Column family Factorys with comparator=UTF8Type
    and default_validation_class=UTF8Type and key_validation_class=UTF8Type
    and column_metadata=
    [
    {column_name: Id_Factory, validation_class: LongType,index_type: KEYS},
    {column_name: Factory_Name, validation_class: UTF8Type},
    {column_name: Factory_Address, validation_class: UTF8Type},
    {column_name: Factory_Tel, validation_class: UTF8Type},
    {column_name: Factory_Type, validation_class: UTF8Type,index_type: KEYS},
    {column_name: Factory_People, validation_class: UTF8Type},
    {column_name: Factory_Cooperation, validation_class: UTF8Type, index_type: KEYS},
    {column_name: Factory_City, validation_class: UTF8Type, index_type: KEYS},
    {column_name: rflag, validation_class: UTF8Type, index_type: KEYS}
    ];
```

```
[default@CassandraKS] create column family Factorys with comparator=UTF8Type
...     and default_validation_class=UTF8Type and key_validation_class=UTF8Type
...     and column_metadata=[
...     {column_name: Id_Factory, validation_class: LongType,index_type: KEYS},
...     {column_name: Factory_Name, validation_class: UTF8Type},
...     {column_name: Factory_Address, validation_class: UTF8Type},
...     {column_name: Factory_Tel, validation_class: UTF8Type},
...     {column_name: Factory_Type, validation_class: UTF8Type,index_type: KEYS},
...     {column_name: Factory_People, validation_class: UTF8Type},
...     {column_name: Factory_Cooperation, validation_class: UTF8Type, index_type: KEYS},
...     {column_name: Factory_City, validation_class: UTF8Type, index_type: KEYS}
...     ];
976449a0-2247-11e1-0000-242d50cf1fbd
Waiting for schema agreement...
... schemas agree across the cluster
[default@CassandraKS]
```

5)、验证列簇（Column family）：Products 和 Factorys

[default@CassandraKS] List Products;
[default@CassandraKS] List Factorys;

```
[default@CassandraKS] List Products;
Using default limit of 100

0 Row Returned.
Elapsed time: 31 msec(s).
[default@CassandraKS] List Factorys;
Using default limit of 100

0 Row Returned.
Elapsed time: 15 msec(s).
[default@CassandraKS]
```

注：上图说明 Products 和 Factorys 的数据都为空，因为我们刚建立了列簇（Column family）结构，还没有向里面添加数据。

6)、更新列簇（Column family）：Products 和 Factorys

实际上很多情况需要更新到列簇（Column family），加入本例子中，我们需要更新 Products 和 Factorys，让其所有的列全部是 UTF-8 类型，实际上这个类型很实用，建议字段都用该类型，除非有特别需要，比较计算时需要浮点数或是双精度数类型。

更新 Products

```
[default@CassandraKS] Update Column family   Products with comparator=UTF8Type
      and default_validation_class=UTF8Type and key_validation_class=UTF8Type
      and column_metadata=
      [
      {column_name: Id_Product, validation_class: UTF8Type,index_type: KEYS},
      {column_name: Product_Name, validation_class: UTF8Type},
      {column_name: Id_Factory, validation_class: UTF8Type,index_type: KEYS},
```

{column_name: Factory_Name, validation_class: UTF8Type},
{column_name: Product_Type, validation_class: UTF8Type,index_type: KEYS},
{column_name: Product_Place, validation_class: UTF8Type},
{column_name: Product_Price, validation_class: UTF8Type, index_type: KEYS},
{column_name: Product_Count, validation_class: UTF8Type },
{column_name: Product_Income, validation_class: UTF8Type, index_type: KEYS},
{column_name: Product_Batch, validation_class: UTF8Type},
{column_name: rflag, validation_class: UTF8Type, index_type: KEYS}
];

```
[default@CassandraKS] Update Column family  Products with comparator=UTF8Type
...     and default_validation_class=UTF8Type and key_validation_class=UTF8Type
...     and column_metadata=
...     [
...     {column_name: Id_Product, validation_class: UTF8Type,index_type: KEYS},
...     {column_name: Product_Name, validation_class: UTF8Type},
...     {column_name: Id_Factory, validation_class: UTF8Type,index_type: KEYS},
...     {column_name: Factory_Name, validation_class: UTF8Type},
...     {column_name: Product_Type, validation_class: UTF8Type,index_type: KEYS},
...     {column_name: Product_Place, validation_class: UTF8Type},
...     {column_name: Product_Price, validation_class: UTF8Type, index_type: KEYS},
...     {column_name: Product_Count, validation_class: UTF8Type },
...     {column_name: Product_Income, validation_class: UTF8Type, index_type: KEYS},
...     {column_name: Product_Batch, validation_class: UTF8Type}
...     ];
e7a190d0-22f6-11e1-0000-242d50cf1fb6
Waiting for schema agreement...
... schemas agree across the cluster
[default@CassandraKS]
```

更新 Factorys

[default@CassandraKS] **Update** Column family Factorys with comparator=UTF8Type
and default_validation_class=UTF8Type and key_validation_class=UTF8Type
and column_metadata=
[
{column_name: Id_Factory, validation_class: LongType,index_type: KEYS},
{column_name: Factory_Name, validation_class: UTF8Type},
{column_name: Factory_Address, validation_class: UTF8Type},
{column_name: Factory_Tel, validation_class: UTF8Type},
{column_name: Factory_Type, validation_class: UTF8Type,index_type: KEYS},
{column_name: Factory_People, validation_class: UTF8Type},
{column_name: Factory_Cooperation, validation_class: UTF8Type, index_type: KEYS},
{column_name: Factory_City, validation_class: UTF8Type, index_type: KEYS},
{column_name: rflag, validation_class: UTF8Type, index_type: KEYS}
];

```
[default@CassandraKS] update column family Factorys with comparator=UTF8Type
...     and default_validation_class=UTF8Type and key_validation_class=UTF8Type
...     and column_metadata=[
...     {column_name: Id_Factory, validation_class: UTF8Type,index_type: KEYS},
...     {column_name: Factory_Name, validation_class: UTF8Type},
...     {column_name: Factory_Address, validation_class: UTF8Type},
...     {column_name: Factory_Tel, validation_class: UTF8Type},
...     {column_name: Factory_Type, validation_class: UTF8Type,index_type: KEYS},
...     {column_name: Factory_People, validation_class: UTF8Type},
...     {column_name: Factory_Cooperation, validation_class: UTF8Type, index_type: KEYS}
...     {column_name: Factory_City, validation_class: UTF8Type, index_type: KEYS}
...     ];
c4c32c90-22f6-11e1-0000-242d50cf1fb6
Waiting for schema agreement...
... schemas agree across the cluster
```

7)、向列簇（Column family）：Products和Factorys各添加 2 条记录（程序插入方式请参考下文客户端操作实例）

Products 第一条记录

```
[default@CassandraKS] set Products[1][Id_Product] = 1;
[default@CassandraKS] set Products[1][Product_Name] = 'Thinkpad T420 ';
[default@CassandraKS] set Products[1][Id_Factory] = 1;
[default@CassandraKS] set Products[1][Factory_Name] = 'Lenovo Com.Ltd';
[default@CassandraKS] set Products[1][Product_Type] = 'Computer';
[default@CassandraKS] set Products[1][Product_Place] = 'Japan';
[default@CassandraKS] set Products[1][Product_Price] = '$2000';
[default@CassandraKS] set Products[1][Product_Count] = 120;
[default@CassandraKS] set Products[1][Product_Income] = '2011/12/05';
[default@CassandraKS] set Products[1][Product_Batch] = 'A-C';
[default@CassandraKS] set Products[1][rflag] = '1';
```

Products 第二条记录

```
[default@CassandraKS] set Products[2][Id_Product] = 2;
[default@CassandraKS] set Products[2][Product_Name] = 'Thinkpad E500 ';
[default@CassandraKS] set Products[2][Id_Factory] = 1;
[default@CassandraKS] set Products[2][Factory_Name] = 'Lenovo Com.Ltd';
[default@CassandraKS] set Products[2][Product_Type] = 'Computer';
[default@CassandraKS] set Products[2][Product_Place] = 'China';
[default@CassandraKS] set Products[2][Product_Price] = '$3000';
[default@CassandraKS] set Products[2][Product_Count] = 160;
[default@CassandraKS] set Products[2][Product_Income] = '2011/10/15';
[default@CassandraKS] set Products[2][Product_Batch] = 'T-D';
[default@CassandraKS] set Products[2][rflag] = '1';
```

Factorys 第一条记录

```
[default@CassandraKS] set Factorys[1][Id_Factory] = 1;
[default@CassandraKS] set Factorys[1][Factory_Name] = 'Lenovo Tec Com.Ltd';
```

[default@CassandraKS] set Factorys[1][Factory_Address] = '5F 51# Peiking building PK CN';
[default@CassandraKS] set Factorys[2][Factory_Tel] = '0592-680984093';
[default@CassandraKS] set Factorys[1][Factory_Type] = 'A';
[default@CassandraKS] set Factorys[1][Factory_People] = 'Mr.Liu';
[default@CassandraKS] set Factorys[1][Factory_Cooperation] = 'A';
[default@CassandraKS] set Factorys[1][Factory_City] = 'BEIJING';
[default@CassandraKS] set Factorys [1][rflag] = '1';

Factorys 第二条记录
[default@CassandraKS] set Factorys[2][Id_Factory] = 2;
[default@CassandraKS] set Factorys[2][Factory_Name] = 'Reach Tec Com.Ltd';
[default@CassandraKS] set Factorys[2][Factory_Address] = '5F 51# Software Park Guanri road Xiamen FJ CN';
[default@CassandraKS] set Factorys[2][Factory_Tel] = '010-6840984093';
[default@CassandraKS] set Factorys[2][Factory_Type] = 'C';
[default@CassandraKS] set Factorys[2][Factory_People] = 'Mr.Xie';
[default@CassandraKS] set Factorys[2][Factory_Cooperation] = 'A';
[default@CassandraKS] set Factorys[2][Factory_City] = 'XIAMEN';
[default@CassandraKS] set Factorys [2][rflag] = '1';

8)、重新验证列簇（Column family）：Products 和 Factorys

[default@CassandraKS] List Products;

```
[default@CassandraKS] list Products;
Using default limit of 100
-------------------
RowKey: 2
=> (column=Factory_Name, value=Lenovo Tec Com.Ltd, timestamp=1323426565046000)
=> (column=Id_Factory, value=1, timestamp=1323426565015000)
=> (column=Id_Product, value=2, timestamp=1323426564984000)
=> (column=Product_Batch, value=T-D, timestamp=1323426566312000)
=> (column=Product_Count, value=160, timestamp=1323426565109000)
=> (column=Product_Income, value=2011/10/15, timestamp=1323426565125000)
=> (column=Product_Name, value=Thinkpad E500 , timestamp=1323426565015000)
=> (column=Product_Place, value=China, timestamp=1323426565078000)
=> (column=Product_Price, value=$3000, timestamp=1323426565093000)
=> (column=Product_Type, value=Computer, timestamp=1323426565062000)
-------------------
RowKey: 1
=> (column=Factory_Name, value=Lenovo Tec Com.Ltd, timestamp=1323426629140000)
=> (column=Id_Factory, value=1, timestamp=1323426629109000)
=> (column=Id_Product, value=1, timestamp=1323426629078000)
=> (column=Product_Batch, value=A-C, timestamp=1323426630312000)
=> (column=Product_Count, value=120, timestamp=1323426629187000)
=> (column=Product_Income, value=2011/12/15, timestamp=1323426629203000)
=> (column=Product_Name, value=Thinkpad T420, timestamp=1323426629109000)
=> (column=Product_Place, value=Japan, timestamp=1323426629156000)
=> (column=Product_Price, value=$2000, timestamp=1323426629171000)
=> (column=Product_Type, value=Computer, timestamp=1323426629140000)

2 Rows Returned.
Elapsed time: 47 msec(s).
[default@CassandraKS]
```

[default@CassandraKS] List Factorys;

```
[default@CassandraKS] List Factorys;
Using default limit of 100
-------------------
RowKey: 2
=> (column=Factory_Address, value=5F 51# Software Park Guanri road Xiamen FJ CN, timestamp=1323426404968000)
=> (column=Factory_City, value=XIAMEN, timestamp=1323426406218000)
=> (column=Factory_Cooperation, value=A, timestamp=1323426405031000)
=> (column=Factory_Name, value=Reach Tec Com.Ltd, timestamp=1323426404937000)
=> (column=Factory_People, value=Mr.Xie, timestamp=1323426405015000)
=> (column=Factory_Tel, value=0592-680984093, timestamp=1323426531953000)
=> (column=Factory_Type, value=C, timestamp=1323426405000000)
=> (column=Id_Factory, value=2, timestamp=1323426404906000)
-------------------
RowKey: 1
=> (column=Factory_Address, value=5F 51# Peiking building PK CN, timestamp=1323426466671000)
=> (column=Factory_City, value=BEIJING, timestamp=1323426467906000)
=> (column=Factory_Cooperation, value=A, timestamp=1323426466750000)
=> (column=Factory_Name, value=Lenovo Tec Com.Ltd, timestamp=1323426466640000)
=> (column=Factory_People, value=Mr.Liu, timestamp=1323426466734000)
=> (column=Factory_Tel, value=010-6840984093, timestamp=1323426466687000)
=> (column=Factory_Type, value=A, timestamp=1323426466718000)
=> (column=Id_Factory, value=1, timestamp=1323426466625000)

2 Rows Returned.
Elapsed time: 47 msec(s).
[default@CassandraKS]
```

注:可以看出各有两条数据

3.Cassandra二级索引理解和分析

上文提到了“index_type – 列索引的 Key，KEYS 列索引的 Value”，这是 Cassandra 很重要的技术，我们估计没有很好的学习材料，发现很多的开发人员都不能很好的利用 Cassandra 的索引和条件查询以及范围查询。

Cassandra 在建立列簇（Column family）是默认有一个索引（关于索引的概念请参考附录），那就是 Key，每行的 Key 标记就是列簇（Column family）的第一索引，那么当我们需要使用其他列的值作为查询条件时，势必会出现一些其他的问题，那就是如何定位到列，在 Cassandra 中，对列值（column values）建立的索引叫做“二级索引”，它与列簇（Column family）中对 Key 的索引不同。二级索引允许我们对列值进行条件查询以及范围查询，并且在读取和写入的时候不会引起操作阻塞。

Cassandra 从 0.7 版本开始支持 KEYS，即是二级索引的定义方式。定义格式如下 {column_name: Id_Product, validation_class: LongType,index_type: KEYS}，这是给列 Id_Product 定义了二级索引，让程序可以把 Id_Product 作为查询条件。查询的格式如下：

Cassandra-Cli格式：（关于Java格式参考客户端操作）

[default@ CassandraKS] get Product s where Id_Product = <?>;

其中？代表具体操作时某个值，比如 2 或是 3 或是其他的。

比如执行:

[default@ CassandraKS] get Products where Id_Product = 1;

```
[default@CassandraKS] get Product s where Id_Product = 1;
Command not found: `get Product s where Id_Product = 1;`. Type 'help;' or '?' for help.
[default@CassandraKS] get Products where Id_Product = 1;
-------------------
RowKey: 1
=> (column=Factory_Name, value=Lenovo Tec Com.Ltd, timestamp=1323426629140000)
=> (column=Id_Factory, value=1, timestamp=1323426629109000)
=> (column=Id_Product, value=1, timestamp=1323426629078000)
=> (column=Product_Batch, value=A-C, timestamp=1323426630312000)
=> (column=Product_Count, value=120, timestamp=1323426629187000)
=> (column=Product_Income, value=2011/12/15, timestamp=1323426629203000)
=> (column=Product_Name, value=Thinkpad T420, timestamp=1323426629109000)
=> (column=Product_Place, value=Japan, timestamp=1323426629156000)
=> (column=Product_Price, value=$2000, timestamp=1323426629171000)
=> (column=Product_Type, value=Computer, timestamp=1323426629140000)

1 Row Returned.
Elapsed time: 47 msec(s).
[default@CassandraKS]
```

Cassandra-cli 只返回了 Id_Product = 1 得数据。

如果多个条件呢？我们是否可以这么理解，多条件就是在 where 语句后面加入 and，然后再加入其他条件呢？格式确实是如此，但是不能像关系型数据库那么强大，支持很多计算规则，比如 between、or、like 等等。Cassandra 到目前的版本，二级索引确实起到了很重要的作用，但是并不能支持很复杂的条件查询，但是基本的运算还是可以满足的，比如：“=”、“>”、“>=”、“<”、“<=”,而这些运算符的支持已经满足了绝大部分程序的需要，加上 Cassandra 客户端 API 的帮助，基本可以实现应用中绝大部分的查询需要，比如过滤、精确定位、范围查询等。

一个列簇（Column family）可以有多个二级索引列，上面的“产品”为例，下表中，红色字体部分都是二级索引，都可以作为条件帮助数据的查询。

字段名称	数据类型（附录）	中文意义	允许为空	是否索引
Id_Product	**LongType**	**产品 Id**	**Not NULL**	**KEYS**
Product_Name	UTF8Type	产品名称	Not NULL	
Id_Factory	**LongType**	**生产厂家 Id**	**Not NULL**	**KEYS**
Factory_Name	UTF8Type	生产厂家名称	Not NULL	
Product_Type	**UTF8Type**	**产品类型**	**Not NULL**	**KEYS**
Product_Place	UTF8Type	产品原产地	NULL	
Product_Price	**UTF8Type**	**产品价格**	**Not NULL**	**KEYS**
Product_Count	LongType	产品数量	Not NULL	
Product_Income	**UTF8Type**	**进货日期**	**NULL**	**KEYS**

Product_Batch	UTF8Type	进货批次	NULL	

比如我们洗完查询产品类型为“Computer”而且生产商是“Lenovo Com.Ltd”的所有产品，语句格式如下：

[default@ CassandraKS] get Products where **Product_Type** = 'Computer' and **Id_Factory** = **1**;

```
[default@CassandraKS] get Products where Product_Type =  'Computer' and Id_Factory = 1;
-------------------
RowKey: 2
=> (column=Factory_Name, value=Lenovo Tec Com.Ltd, timestamp=1323426565046000)
=> (column=Id_Factory, value=1, timestamp=1323426565015000)
=> (column=Id_Product, value=2, timestamp=1323426564984000)
=> (column=Product_Batch, value=T-D, timestamp=1323426566312000)
=> (column=Product_Count, value=160, timestamp=1323426565109000)
=> (column=Product_Income, value=2011/10/15, timestamp=1323426565125000)
=> (column=Product_Name, value=Thinkpad E500 , timestamp=1323426565015000)
=> (column=Product_Place, value=China, timestamp=1323426565078000)
=> (column=Product_Price, value=$3000, timestamp=1323426565093000)
=> (column=Product_Type, value=Computer, timestamp=1323426565062000)
-------------------
RowKey: 1
=> (column=Factory_Name, value=Lenovo Tec Com.Ltd, timestamp=1323426629140000)
=> (column=Id_Factory, value=1, timestamp=1323426629109000)
=> (column=Id_Product, value=1, timestamp=1323426629078000)
=> (column=Product_Batch, value=A-C, timestamp=1323426630312000)
=> (column=Product_Count, value=120, timestamp=1323426629187000)
=> (column=Product_Income, value=2011/12/15, timestamp=1323426629203000)
=> (column=Product_Name, value=Thinkpad T420, timestamp=1323426629109000)
=> (column=Product_Place, value=Japan, timestamp=1323426629156000)
=> (column=Product_Price, value=$2000, timestamp=1323426629171000)
=> (column=Product_Type, value=Computer, timestamp=1323426629140000)

2 Rows Returned.
Elapsed time: 391 msec(s).
[default@CassandraKS]
```

如果在原来的查询条件基础上加上一个进货日期是 2011/12/15，那么查询出来的结果所要表达的大意就是“2011/12/15 从联想进了多少计算机类型的产品”。

格式如下：

[default@ CassandraKS] get Products where

Product_Type = 'Computer' and Id_Factory = 1 and Product_Income = '2011/12/15';

```
[default@CassandraKS] get Products where
...      Product_Type = 'Computer'
...      and Id_Factory = 1
...      and Product_Income =  '2011/12/15';
-------------------
RowKey: 1
=> (column=Factory_Name, value=Lenovo Tec Com.Ltd, timestamp=1323426629140000)
=> (column=Id_Factory, value=1, timestamp=1323426629109000)
=> (column=Id_Product, value=1, timestamp=1323426629078000)
=> (column=Product_Batch, value=A-C, timestamp=1323426630312000)
=> (column=Product_Count, value=120, timestamp=1323426629187000)
=> (column=Product_Income, value=2011/12/15, timestamp=1323426629203000)
=> (column=Product_Name, value=Thinkpad T420, timestamp=1323426629109000)
=> (column=Product_Place, value=Japan, timestamp=1323426629156000)
=> (column=Product_Price, value=$2000, timestamp=1323426629171000)
=> (column=Product_Type, value=Computer, timestamp=1323426629140000)

1 Row Returned.
Elapsed time: 31 msec(s).
[default@CassandraKS]
```

重要备注：如果列簇（Column family）已经有数据，然后你又重建了列簇（Column family）的索引，那么重建之前的那些数据一样不能索引，只能写个程序（比如读取一条，再以同样的 key 插入数据，也就是数据覆盖一遍）重新插入这些数据才能实现索引那些旧的数据。

4.Cassandra客户端操作实例

接下来，我们通过客户端 API 实现 Cassandra 具体使用过程。

4.1. Cassandra Java客户端API如何链接数据库

连接池+分布式群集架构方式是 Cassandra 运用的最大特点，群集我们已经在其他文章中提及了，这里主要讲解一下连接池的运用。

A、下面四句语句表示了 Cassandra 客户端需要的基本参数。

（代码属于: `TestCassandraClient.java`）

```
public static String hosts = "127.0.0.1:9160";
public static String clusterName = "LKXCluster";
public static String keyspaceName = "CassandraKS";
```

```
public static String columnFamily = "Products";
```

B、通过一个 Class 装载这四个参数

(代码属于: CassandraParas.java)

```
package com.reach.leekexi.cassandra.hecter;
/**
 * Cassandra 初始化需要的参数
 * @author 李克喜
 * 2011/11/16 AM09:58
 */
public class CassandraParas {
    //格式"192.168.0.1:9160,192.168.0.2:9160"每一个IP和一个端口为一组
    public String hosts = "";
    //群集名称
    public String clusterName = "";
    //键空间 == 可以理解为数据库
    public String keyspaceName;
    //列家族，也就是所有列的Parent;,可以理解为一个表
    public String columnFamily;

    public String getHosts() {
        return hosts;
    }
    public void setHosts(String hosts) {
        this.hosts = hosts;
    }
    public String getClusterName() {
        return clusterName;
    }
    public void setClusterName(String clusterName) {
        this.clusterName = clusterName;
    }
    public String getKeyspaceName() {
        return keyspaceName;
    }
    public void setKeyspaceName(String keyspaceName) {
        this.keyspaceName = keyspaceName;
    }

    public String getColumnFamily() {
        return columnFamily;
    }
}
```

```
    public void setColumnFamily(String columnFamily) {
        this.columnFamily = columnFamily;
    }

}
```

C、通过一个调用装载：

(代码属于: TestCassandraClient.java)

```
//参数装载
public static void main(String[] args) {
    //参数装载
    CassandraParas ps = new CassandraParas();
    ps.setHosts(hosts); //注解数据
    ps.setClusterName(clusterName);//群集名称
    ps.setKeyspaceName(keyspaceName);//键空间
    ps.setColumnFamily(columnFamily);//列族
}
```

D、通过构造一个客户端实例，实现链接

(代码属于: TestCassandraClient.java)

```
CassandraClient cc = new CassandraClient(ps);
```

`TestCassandraClient.java` 的全部内容如下：

```
public class TestCassandraClient {
    public static String hosts = "192.168.0.8:9160,192.168.0.9:9160";
    public static String clusterName = "LKXCluster";
    public static String keyspaceName = "indexKS";
    public static String columnFamily = "Users";

    @SuppressWarnings("rawtypes")
    public static void main(String[] args) {
        //参数装载
        CassandraParas ps = new CassandraParas();
        ps.setHosts(hosts); //注解数据
        ps.setClusterName(clusterName);//群集名称
        ps.setKeyspaceName(keyspaceName);//键空间
        ps.setColumnFamily(columnFamily);//列族
        CassandraClient cc = new CassandraClient(ps);
    }
```

```
}
```

E、关于 CassandraClient.java 这里不做详述，已经做过比较完整的封装了，具体可以参考 SVN 上的 HecterExample 工程。这里只列出够着连接池的基本过程：

```
public class CassandraClient {

    public String KEYSPACE = ""; // 相当于数据库
    public String CF_NAME = ""; // 相当于表
    public String ROWKEY = "";
    public String COLUMN_NAME = ""; // 一个列的名称
    public String COLUMN_VALUE = ""; // 一个列的值
    public StringSerializer serializer = StringSerializer.get(); //序列
化字符串
    public String hosts = ""; //群集主机列表
    public String clusterName= ""; //群集名称
    private CassandraHostConfigurator cc;//群集主机设置
    private Cluster c;//群集
    private Keyspace keyspace;//键空间变量

    /**
     * 提供cassandra参数
     * @param cps
     */
    public CassandraClient(CassandraParas cps) {
        KEYSPACE = cps.getKeyspaceName();
        CF_NAME = cps.getColumnFamily();
        hosts = cps.getHosts();
        clusterName = cps.getClusterName();
        cc = new CassandraHostConfigurator(hosts);
        c = getOrCreateCluster(clusterName, cc);
        keyspace = createKeyspace(KEYSPACE, c);
    }
}
```

这样我们就完成了数据库连接池的建立。

4.2. 通过java向Products列簇添加一条记录

A、这里只给出调用代码，并没有给出详细的实现如何插入的过程，因为那些都是包装成 API 了。

(代码属于: TestCassandraClient.java)

```
String key = "3";//后面说怎么做自动递增的 key
String[] columnNames =
{"Id_Product","Product_Name","Id_Factory","Factory_Name","Product_Type","Product_Place","Product_Price","Product_Count","Product_Income","Product_Batch"};
String[] columnValues = {"3","乐 PAD A107","1","联想","移动通讯","天津","890.00","300","2011/10/12","T-C"};
boolean inf = cc.insertMultiColumn(key, columnNames, columnValues);
if(inf)
{
    Map map = new HashMap();
    //插入如果返回 true，则测试读取数据
    map = cc.getMultiColumn(key, columnNames);
    for(int i = 0;i < map.size() ;i++)
        System.out.println(columnNames[i] + " : " +
        map.get(columnNames[i]));
}
```

B、运行结果：

Eclipse 控制台：

```
14:27:36,921  INFO CassandraHostRetryService:37 -
14:27:37,000  INFO JmxMonitor:54 - Registering JMX
Id_Product : 3
Product_Name : 乐PAD A107
Id_Factory : 1
Factory_Name : 联想
Product_Type : 移动通讯
Product_Place : 天津
Product_Price : 890.00
Product_Count : 300
Product_Income : 2011/10/12
Product_Batch : T-C
```

C、Cassandra-cli 客户端执行：

[default@CassandraKS] List Products;

```
[default@CassandraKS] list Products;
Using default limit of 100
-------------------
RowKey: 3
=> (column=Factory_Name, value=联想, timestamp=1323498457078002)
=> (column=Id_Factory, value=1, timestamp=1323498457078001)
=> (column=Id_Product, value=3, timestamp=1323498457000000)
=> (column=Product_Batch, value=T-C, timestamp=1323498457093005)
=> (column=Product_Count, value=300, timestamp=1323498457093003)
=> (column=Product_Income, value=2011/10/12, timestamp=1323498457093004)
=> (column=Product_Name, value=乐PAD A107, timestamp=1323498457078000)
=> (column=Product_Place, value=天津, timestamp=1323498457093001)
=> (column=Product_Price, value=890.00, timestamp=1323498457093002)
=> (column=Product_Type, value=移动通讯, timestamp=1323498457093000)
-------------------
RowKey: 2
=> (column=Factory_Name, value=Lenovo Com.Ltd, timestamp=1323498430562000)
=> (column=Id_Factory, value=1, timestamp=1323498430546000)
=> (column=Id_Product, value=2, timestamp=1323498430500000)
=> (column=Product_Batch, value=T-D, timestamp=1323498431734000)
=> (column=Product_Count, value=160, timestamp=1323498430625000)
=> (column=Product_Income, value=2011/10/15, timestamp=1323498430640000)
=> (column=Product_Name, value=Thinkpad E500 , timestamp=1323498430531000)
=> (column=Product_Place, value=China, timestamp=1323498430593000)
=> (column=Product_Price, value=$3000, timestamp=1323498430609000)
=> (column=Product_Type, value=Computer, timestamp=1323498430578000)
-------------------
RowKey: 1
=> (column=Factory_Name, value=Lenovo Com.Ltd, timestamp=1323498381656000)
=> (column=Id_Factory, value=1, timestamp=1323498381640000)
=> (column=Id_Product, value=1, timestamp=1323498381562000)
=> (column=Product_Batch, value=A-C, timestamp=1323498383906000)
=> (column=Product_Count, value=120, timestamp=1323498381718000)
=> (column=Product_Income, value=2011/12/05, timestamp=1323498381734000)
=> (column=Product_Name, value=Thinkpad T420 , timestamp=1323498381625000)
=> (column=Product_Place, value=Japan, timestamp=1323498381687000)
=> (column=Product_Price, value=$2000, timestamp=1323498381703000)
=> (column=Product_Type, value=Computer, timestamp=1323498381671000)

3 Rows Returned.
Elapsed time: 62 msec(s).
[default@CassandraKS]
```

这证明数据已经存入 Cassandra 数据库了。

新增数据和修改数据执行的代码其实是一样的，只要 key 一致，那么新的数据就会覆盖旧的数据。

4.3. 通过java获得Products列簇所有记录

(代码属于: TestCassandraClient.java)

```
String[] columnNames = {"Id_Product","Product_Name","Id_Factory",
                "Factory_Name","Product_Type","Product_Place",
                "Product_Price","Product_Count","Product_Income",
                "Product_Batch"};

//查询所有的数据开始
String[] exColumnNames =  {"rflag"};
String[] expressions =    {"="};
String[] exColumnvalues = {"1"};
Map rowValues  = cc.getMultiColumnByIndex(columnNames, exColumnNames,
exColumnvalues, expressions);
for(int i = rowValues.size()-1;i >= 0;i--)
{
    Map columnValues = (Map) rowValues.get(i);
    for(int j = 0;j < columnValues.size() ;j++)
        System.out.print("[" + columnNames[j] + " = " +
columnValues.get(columnNames[j]) + "],");
    System.out.println("");
}
//查询所有的数据结束
```

Eclipse 控制台运行结果：

[Id_Product = 1],[Product_Name = Thinkpad T420],[Id_Factory = 1],[Factory_Name = Lenovo Com.Ltd],[Product_Type = Computer],[Product_Place = Japan],[Product_Price = $2000],[Product_Count = 120],[Product_Income = 2011/12/05],[Product_Batch = A-C]
[Id_Product = 2],[Product_Name = Thinkpad E500],[Id_Factory = 1],[Factory_Name = Lenovo Com.Ltd],[Product_Type = Computer],[Product_Place = China],[Product_Price = $3000],[Product_Count = 160],[Product_Income = 2011/10/15],[Product_Batch = T-D]
[Id_Product = 3],[Product_Name = 乐 PAD A107],[Id_Factory = 1],[Factory_Name = 联想],[Product_Type = 移动通讯],[Product_Place = 天津],[Product_Price = 890.00],[Product_Count = 300],[Product_Income = 2011/10/12],[Product_Batch = T-C],

程序中出现了 rflag，前文我们一直没有提及，最主要的问题是怕说不清楚，所以放在这里说明，rflag 也是二级索引，而且值都默认为 1，在插入每一行数据时都给他赋值 1，所以要查询所有数据时加入这个条件和没有加入这个条件结果基本一致。那么为什么要加入呢？问题在于 Cassandra 在删除数据行后，key 保留着，所以虽然某一行的数据已经不存在了，但是 key 还在，所以当你不加入过滤时，查出的数据有很多可能为 null 的行。

4.4. 通过java获得Products列簇部分记录

(代码属于: TestCassandraClient.java)

```
String[] columnNames = {"Id_Product","Product_Name","Id_Factory",
                "Factory_Name","Product_Type","Product_Place",
                "Product_Price","Product_Count","Product_Income",
                "Product_Batch"};

//查询开始
//查询产品表返回产品类型是“Computer”的数据
String[] exColumnNames =  {"rflag","Product_Type"};
String[] expressions =    {"=","="};
String[] exColumnvalues = {"1","Computer"};
Map rowValues  = cc.getMultiColumnByIndex(columnNames, exColumnNames,
exColumnvalues, expressions);
for(int i = rowValues.size()-1;i >= 0;i--)
{
    Map columnValues = (Map) rowValues.get(i);
    for(int j = 0;j < columnValues.size() ;j++)
    System.out.print("[" + columnNames[j] + " = " +
columnValues.get(columnNames[j]) + "],");
    System.out.println("");
}
//查询结束
```

Eclipse 控制台运行结果：

```
[Id_Product = 1],[Product_Name = Thinkpad T420 ],[Id_Factory =
1],[Factory_Name = Lenovo Com.Ltd],[Product_Type =
Computer],[Product_Place = Japan],[Product_Price =
$2000],[Product_Count = 120],[Product_Income =
2011/12/05],[Product_Batch = A-C]
```

```
[Id_Product = 2],[Product_Name = Thinkpad E500 ],[Id_Factory =
1],[Factory_Name = Lenovo Com.Ltd],[Product_Type =
Computer],[Product_Place = China],[Product_Price =
$3000],[Product_Count = 160],[Product_Income =
2011/10/15],[Product_Batch = T-D]
```

可以看出，只是 exColumnNames，expressions，exColumnvalues 这三个变量发生变化，确实如此，只要按矩阵的方式填写条件，那么 Cassandra 客户端的 API 就会叠加条件，并执行查询。

4.5. 通过java多条件下查询Products列簇

(代码属于: TestCassandraClient.java)

```
String[] columnNames = {"Id_Product","Product_Name","Id_Factory",
              "Factory_Name","Product_Type","Product_Place",
              "Product_Price","Product_Count","Product_Income",
              "Product_Batch"};

//查询开始
//查询产品表返回产品类型是"Computer",进货日期大于"2011/10/15"的数据
String[] exColumnNames =  {"rflag","Product_Type","Product_Income"};
String[] expressions =    {"=","=",">"};
String[] exColumnvalues = {"1","Computer","2011/10/15"};
Map rowValues  = cc.getMultiColumnByIndex(columnNames, exColumnNames,
exColumnvalues, expressions);
for(int i = rowValues.size()-1;i >= 0;i--)
{
    Map columnValues = (Map) rowValues.get(i);
    for(int j = 0;j < columnValues.size() ;j++)
    System.out.print("[" + columnNames[j] + " = " +
columnValues.get(columnNames[j]) + "],");
    System.out.println("");
}
//查询结束
Eclipse 控制台运行结果:
```

```
[Id_Product = 1],[Product_Name = Thinkpad T420 ],[Id_Factory =
1],[Factory_Name = Lenovo Com.Ltd],[Product_Type =
Computer],[Product_Place = Japan],[Product_Price =
$2000],[Product_Count = 120],[Product_Income =
2011/12/05],[Product_Batch = A-C]
```

结果很清楚，上一个查询有一个产品的进货日期是“2011/10/15”，而本次查询的条件刚好是进货日期大于“2011/10/15”，不包含进货日期是“2011/10/15”的产品数据，如果还想得到进货日期是“2011/10/15”的产品数据，那么条件变成大于等于(>=)即可.

4.6. 通过java多条件下查询Products列簇，返回部分列

(代码属于: TestCassandraClient.java)

```
//返回
String[] columnNames =
{"Id_Product","Product_Name","Product_Type","Product_Batch"};

//查询开始
//查询产品表返回产品类型是"Computer",进货日期大于"2011/10/15"的数据
String[] exColumnNames =  {"rflag","Product_Type","Product_Income"};
String[] expressions =    {"=","=",">"};
String[] exColumnvalues = {"1","Computer","2011/10/15"};
Map rowValues  = cc.getMultiColumnByIndex(columnNames, exColumnNames,
exColumnvalues, expressions);
for(int i = rowValues.size()-1;i >= 0;i--)
{
    Map columnValues = (Map) rowValues.get(i);
    for(int j = 0;j < columnValues.size() ;j++)
    System.out.print("[" + columnNames[j] + " = " +
columnValues.get(columnNames[j]) + "],");
    System.out.println("");
}
//查询结束
```

Eclipse 控制台运行结果：

```
[Id_Product = 1],[Product_Name = Thinkpad T420 ],[Product_Type =
Computer],[Product_Batch = A-C]
```

4.8. 通过java实现Products列簇分页数据查询

翻页实现的复杂过程已经经过包装，而调用的过程非常简单，只需要提供：

A、上翻还是下翻标记

B、一页返回多少行

C、希望得到那些列数据

(代码属于: TestCassandraClient.java)

```
String[] columnNames =
{"Id_Product","Product_Name","Product_Type","Product_Batch"};

System.out.println("--向下翻页，每页2行，每一页的开始行都是上一页的结束行--");
Map rowValues = cc.getMultiPage(2, columnNames,1);
for(int i = 0;i < rowValues.size();i++)
{
```

```
    Map columnValues = (Map) rowValues.get(i);
    for(int j = 0;j < columnValues.size() ;j++)
        System.out.print("[" + columnNames[j] + " = " +
columnValues.get(columnNames[j]) + "],");
    System.out.println("");
}

System.out.println("--向下翻页，每页2行，每一页的开始行都是上一页的结束行--");
rowValues = cc.getMultiPage(2, columnNames,1);
for(int i = 0;i < rowValues.size();i++)
{
    Map columnValues = (Map) rowValues.get(i);
    for(int j = 0;j < columnValues.size() ;j++)
        System.out.print("[" + columnNames[j] + " = " +
columnValues.get(columnNames[j]) + "],");
    System.out.println("");
}

System.out.println("--向上翻页，每页2行，每一页的结束行都是上一页的开始行--");
rowValues = cc.getMultiPage(2, columnNames,0);
for(int i = 0;i < rowValues.size();i++)
{
    Map columnValues = (Map) rowValues.get(i);
    for(int j = 0;j < columnValues.size() ;j++)
        System.out.print("[" + columnNames[j] + " = " +
columnValues.get(columnNames[j]) + "],");
    System.out.println("");
}
```

Eclipse 控制台运行结果：

```
--向下翻页，每页2行，每一页的开始行都是上一页的结束行--
[Id_Product = 3],[Product_Name = 乐PAD A107],[Product_Type = 移动通讯],[Product_Batch = T-C],
[Id_Product = 2],[Product_Name = Thinkpad E500 ],[Product_Type = Computer],[Product_Batch = T-D],
--向下翻页，每页2行，每一页的开始行都是上一页的结束行--
[Id_Product = 2],[Product_Name = Thinkpad E500 ],[Product_Type = Computer],[Product_Batch = T-D],
[Id_Product = 1],[Product_Name = Thinkpad T420 ],[Product_Type = Computer],[Product_Batch = A-C],
--向上翻页，每页2行，每一页的结束行都是上一页的开始行--
[Id_Product = 3],[Product_Name = 乐PAD A107],[Product_Type = 移动通讯],[Product_Batch = T-C],
[Id_Product = 2],[Product_Name = Thinkpad E500 ],[Product_Type = Computer],[Product_Batch = T-D],
```

4.9. 总结通过java查询Products列簇

执行的效果和 cassandra-cli 的效果一致，而且程序写法简单，适应大部分的查询需要，但是不能满足所有的查询需要。条件的构建和需要获得那些列的数据一样可以自定义。

5.附录

5.1. Cassandra数据类型和CQL对应关系

Internal Type	CQL Name	Description
BytesType	blob	Arbitrary hexadecimal bytes (no validation)
AsciiType	ascii	US-ASCII character string
UTF8Type	text, varchar	UTF-8 encoded string
IntegerType	varint	Arbitrary-precision integer
LongType	int, bigint	8-byte long
UUIDType	uuid	Type 1 or type 4 UUID
DateType	timestamp	Date plus time, encoded as 8 bytes since epoch
BooleanType	boolean	true or false
FloatType	float	4-byte floating point
DoubleType	double	8-byte floating point
DecimalType	decimal	Variable-precision decimal
CounterColumnType	counter	Distributed counter value (8-byte long)

5.2. 数据库索引的概念

[基本概念的解释来自 baidu 百科，这里针对 NQL 做了基本解释]

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。数据库索引好比是一本书前面的目录，能加快数据库的查询速度。

例如这样一个查询：get Products where Id_Products = 44。如果没有索引，必须遍历整个列簇（Column family），直到 Id_Products 等于 44 的这一组被找到为止；有了索引之后(必须是在 Id_Products 这一列上建立的索引)，直接在索引里面找 44（也就是在 Id_Products 这一列找），就可以得知这一行的位置，也就是找到了这一行。可见，索引是用来定位的。

5.3. 本文中使用到的Java文件

A、需要的 jar 包

包含 2 部分，第一部分是 Hector 的 Lib 目录下的所有 jar 文件；第二部分是来自 Cassandra Lib 目录下的所有 jar 文件。合并之后如下图：

至于版本问题，可以自行选择互相对应 Hector 和 Cassandra

比如本文的中采用的 Hector 是 1.0.1 版本，而 Cassandra 是 1.0.2 版本。

名称	大小	类型
uuid-3.2.0.jar	14 KB	Executable Jar File
speed4j-0.9.jar	26 KB	Executable Jar File
snappy-java-1.0.4.1.jar	973 KB	Executable Jar File
snakeyaml-1.6.jar	227 KB	Executable Jar File
slf4j-log4j12-1.6.1.jar	10 KB	Executable Jar File
slf4j-api-1.6.1.jar	25 KB	Executable Jar File
servlet-api-2.5-20081211.jar	131 KB	Executable Jar File
reachCC.jar	11 KB	Executable Jar File
log4j-1.2.16.jar	471 KB	Executable Jar File
libthrift-0.6.jar	289 KB	Executable Jar File
libthrift-0.6.1.jar	289 KB	Executable Jar File
json-simple-1.1.jar	16 KB	Executable Jar File
jline-0.9.94.jar	86 KB	Executable Jar File
jamm-0.2.5.jar	6 KB	Executable Jar File
jackson-mapper-asl-1.4.0.jar	378 KB	Executable Jar File
jackson-core-asl-1.4.0.jar	147 KB	Executable Jar File
high-scale-lib-1.1.2.jar	94 KB	Executable Jar File
hector-core-1.0-1-tests.jar	149 KB	Executable Jar File
hector-core-1.0-1-sources.jar	244 KB	Executable Jar File
hector-core-1.0-1.jar	761 KB	Executable Jar File
guava-r09.jar	1,118 KB	Executable Jar File
guava-r08.jar	1,087 KB	Executable Jar File
FastInfoset-1.2.2.jar	285 KB	Executable Jar File
concurrentlinkedhashmap-lru-1.2.jar	52 KB	Executable Jar File
compress-lzf-0.8.4.jar	25 KB	Executable Jar File
commons-pool-1.5.3.jar	94 KB	Executable Jar File
commons-lang-2.4.jar	256 KB	Executable Jar File
commons-codec-1.2.jar	30 KB	Executable Jar File
commons-cli-1.1.jar	36 KB	Executable Jar File
cassandra-thrift-1.0.0.jar	834 KB	Executable Jar File
avro-1.4.0-sources-fixes.jar	270 KB	Executable Jar File
avro-1.4.0-fixes.jar	583 KB	Executable Jar File
apache-cassandra-thrift-1.0.2.jar	842 KB	Executable Jar File
apache-cassandra-clientutil-1.0.2.jar	32 KB	Executable Jar File
apache-cassandra-1.0.2.jar	1,744 KB	Executable Jar File
antlr-3.2.jar	1,883 KB	Executable Jar File

B、CassandraParas.java

```
/**
 * Cassandra 初始化需要的参数
 * @author 李克喜
 * 2011/09/11 AM09:58
 */
public class CassandraParas {


    public String hosts = ""; //格式"192.168.0.1:9160,192.168.0.2:9160" 每一个IP和一个端口为一组
    public String clusterName = "";//
    public String keyspaceName; //键空间 == 可以理解为数据库
    public String columnFamily; //列家族，也就是所有列的Parent;,可以理解为一个表

    public String getHosts() {
        return hosts;
    }
    public void setHosts(String hosts) {
        this.hosts = hosts;
    }
    public String getClusterName() {
        return clusterName;
    }
    public void setClusterName(String clusterName) {
        this.clusterName = clusterName;
    }
    public String getKeyspaceName() {
        return keyspaceName;
    }
    public void setKeyspaceName(String keyspaceName) {
        this.keyspaceName = keyspaceName;
    }

    public String getColumnFamily() {
        return columnFamily;
    }
    public void setColumnFamily(String columnFamily) {
        this.columnFamily = columnFamily;
    }
}
```

C、CassandraClient.java

```
import static me.prettyprint.hector.api.factory.HFactory.createKeyspace;
import static me.prettyprint.hector.api.factory.HFactory.getOrCreateCluster;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import me.prettyprint.cassandra.model.IndexedSlicesQuery;
import me.prettyprint.cassandra.serializers.StringSerializer;
import me.prettyprint.cassandra.service.CassandraHostConfigurator;
import me.prettyprint.hector.api.beans.OrderedRows;
import me.prettyprint.hector.api.beans.Row;
import me.prettyprint.hector.api.factory.HFactory;
import me.prettyprint.hector.api.query.QueryResult;
import me.prettyprint.hector.api.query.RangeSlicesQuery;

/**
 * Cassandra数据操作封装改造Hector之后的API界面
 * @author 李克喜
 * 2011/09/11 AM09:58
 */
public class CassandraClient extends HectorBaseApi{

    /**
     * 提供cassandra参数
     * @param cps
     */
    public CassandraClient(CassandraParas cps) {
        KEYSPACE = cps.getKeyspaceName();
        CF_NAME = cps.getColumnFamily();
        hosts = cps.getHosts();
        clusterName = cps.getClusterName();
        cc = new CassandraHostConfigurator(hosts);
        c = getOrCreateCluster(clusterName, cc);
        keyspace = createKeyspace(KEYSPACE, c);
    }
```

```
    /**
     * 添加一个字段和数据
     * @param key
     * @param columnName
     * @param columnValue
     * @return boolean
     */
    public boolean insertOneColumn(String key,String columnName,String columnValue) {

        boolean insertFlag = false;
        ROWKEY = key;
        COLUMN_NAME = columnName;
        COLUMN_VALUE = columnValue;
        insert(ROWKEY,COLUMN_VALUE,StringSerializer.get());

        String tempValue = get(ROWKEY, StringSerializer.get());
        if(tempValue.equals(COLUMN_VALUE))
        {
            insertFlag= true;
        }

        return insertFlag;
    }

    /**
     * 得到一个字段的值
     * @param key
     * @param columnName
     * @return String
     */
    public String getOneColumn(String key,String columnName)
    {
        ROWKEY = key;
        COLUMN_NAME = columnName;
        String tempValue = get(ROWKEY, StringSerializer.get());
        return tempValue;
    }

    /**
     * 删除一个字段
```

```
     * @param key
     * @param columnName
     * @return boolean
     */
    public boolean deleteOneColumn(String key,String columnName)
    {
        boolean deleteFlag = false;
        ROWKEY = key;
        COLUMN_NAME = columnName;
        delete(StringSerializer.get(),ROWKEY);
        String tempValue = get(ROWKEY, StringSerializer.get());
        if(tempValue == null)
        {
            deleteFlag= true;
        }
        return deleteFlag;
    }

    /**
     * 添加多个字段和数据，支持事务模式
     * @param key
     * @param columnName
     * @param columnValue
     * @return boolean
     */
    public boolean insertMultiColumn(String key,String[]
columnName,String[] columnValue) {

        boolean insertFlag = false;
        ROWKEY = key;
        if(columnName.length != columnValue.length)
            return false;
        int block_i = 0;
        for(int i = 0;i < columnName.length ;i++)
        {
            COLUMN_NAME = columnName[i];
            COLUMN_VALUE = columnValue[i];
            insert(ROWKEY,COLUMN_VALUE,StringSerializer.get());
            String tempValue = get(ROWKEY, StringSerializer.get());

            if(tempValue.equals(COLUMN_VALUE))
            {
```

```
                insertFlag= true;
            }
            else
            {
                block_i = i;
                insertFlag= false;
                break;
            }
        }
        //回退，支持事务
        if(!insertFlag)
        {
            for(int i = 0;i < block_i ;i++)
            {
                COLUMN_NAME = columnName[i];
                deleteOneColumn(ROWKEY,COLUMN_NAME);
            }
        }

        return insertFlag;
    }

    /**
     * 得到多个字段的值
     * @param key
     * @param columnName - 欲获得数据的列名称
     * @return Map
     */
    @SuppressWarnings({ "rawtypes", "unchecked" })
    public Map getMultiColumn(String key,String[] columnName)
    {
        Map columnValues = new HashMap();
        ROWKEY = key;
        for(int i = 0;i < columnName.length ;i++)
        {
            COLUMN_NAME = columnName[i];
            String tempValue = get(ROWKEY, StringSerializer.get());
            columnValues.put(COLUMN_NAME, tempValue);
        }
        return columnValues;
    }

```

```
    /**
     * 删除多个字段
     * @param key
     * @param columnName - 欲删除的列的名称
     * @return boolean
     */
    @SuppressWarnings({ "rawtypes", "unchecked" })
    public boolean deleteMultiColumn(String key,String[] columnName)
    {
        boolean deleteFlag = false;
        ROWKEY = key;
        Map map = new HashMap();
        for(int i = 0;i < columnName.length ;i++)
        {
            COLUMN_NAME = columnName[i];
            map.put(COLUMN_NAME, get(ROWKEY, StringSerializer.get()));
            delete(StringSerializer.get(),ROWKEY);
            String tempValue = get(ROWKEY, StringSerializer.get());

            if(tempValue == null)
            {
                deleteFlag= true;
            }
            else
            {
                deleteFlag = false;
                break;
            }
        }
        //事务处理回退删除
        if(!deleteFlag)
        {
            for(int i = 0; i < map.size();i++)
            {
                COLUMN_NAME = columnName[i];
                COLUMN_VALUE = (String) map.get(COLUMN_NAME);
                insert(ROWKEY,COLUMN_VALUE,StringSerializer.get());
            }
        }
        return deleteFlag;
    }

```

```
    /**
     * 二级索引查询,适合范围查询和精确查询
     * @param getColumnNames - 获得那些列的值
     * @param exColumnNames - 条件列
     * @param exColumnvalues - 条件列的值
     * @param expressions - 表达式
     */
    @SuppressWarnings({ "rawtypes", "unchecked" })
    public Map getMultiColumnByIndex(String[] ColumnNames,String[]exColumnNames, String[] exColumnvalues ,String[] expressions)
    {
        Map rowValues = null;
        try
        {
            StringSerializer ss = StringSerializer.get();
            IndexedSlicesQuery<String, String, String> indexedSlicesQuery = HFactory.createIndexedSlicesQuery(keyspace, ss, ss, ss);
            indexedSlicesQuery.setColumnNames(ColumnNames);

            if(exColumnNames != null)
            for(int i = 0 ;i < exColumnNames.length ;i++)
            {
                if(expressions[i].equals("="))
    indexedSlicesQuery.addEqualsExpression(exColumnNames[i], exColumnvalues[i]);
                else if (expressions[i].equals(">="))
    indexedSlicesQuery.addGteExpression(exColumnNames[i], exColumnvalues[i]);
                else if (expressions[i].equals(">"))
    indexedSlicesQuery.addGtExpression(exColumnNames[i], exColumnvalues[i]);
                else if (expressions[i].equals("<="))
    indexedSlicesQuery.addLteExpression(exColumnNames[i], exColumnvalues[i]);
```

```
                else if (expressions[i].equals("<"))

    indexedSlicesQuery.addLtExpression(exColumnNames[i],
exColumnvalues[i]);
            }
            else
                indexedSlicesQuery.addEqualsExpression("","");

            indexedSlicesQuery.setColumnFamily(CF_NAME);
            indexedSlicesQuery.setStartKey("");
            QueryResult<OrderedRows<String, String, String>> result =
indexedSlicesQuery.execute();

            OrderedRows<String, String, String> rd = result.get();

            List l = new ArrayList();
            l = rd.getList();
            rowValues = new HashMap();
            int rowf = 0;
            for (Iterator iter = l.iterator(); iter.hasNext();) {
                Map columnValues = new HashMap();
                Row<String, String, String> row = (Row<String, String,
String> )iter.next();
                for(int i = 0;i < ColumnNames.length;i++)
                {
                    String cv = (String)
row.getColumnSlice().getColumnByName(ColumnNames[i]).getValue();
                    //System.out.println(ColumnNames[i] + ":" + cv);
                    columnValues.put(ColumnNames[i], cv);
                }
                rowValues.put(rowf, columnValues);
                rowf++;
            }
        }catch(Exception ex)
        {
            ex.printStackTrace();
        }

        return rowValues;
    }

    Row<String,String,String> lastRow = null;
```

```
    /**
     * 分页查询
     * @param rcount - 一页取多少行
     * @param ColumnNames - 每行取得字段名称
     * @param nextOrpre - 1 = 向下翻页 , 0 - 向上翻页
     * @return
     */
    @SuppressWarnings({ "unchecked", "rawtypes" })
    public Map getMultiPage(int rcount,String[] ColumnNames,int nextOrpre)
    {
        Map rowValues = null;
        try
        {
            StringSerializer ss = StringSerializer.get();
            RangeSlicesQuery<String, String, String> rangeSlicesQuery = HFactory.createRangeSlicesQuery(keyspace, ss, ss, ss);
            rangeSlicesQuery.setColumnFamily(CF_NAME);
            rangeSlicesQuery.setRange("", "", false, 3);
            rangeSlicesQuery.setColumnNames(ColumnNames);

            if(lastRow == null)
            {
                rangeSlicesQuery.setKeys("", "");
            }
            else if(nextOrpre == 1 && lastRow != null) //1 = 向下翻页 , 0 - 向上翻页
            {
                rangeSlicesQuery.setKeys(lastRow.getKey(), "");
            }
            else if(nextOrpre == 0 && lastRow != null) //1 = 向下翻页 , 0 - 向上翻页
            {
                rangeSlicesQuery.setKeys("", lastRow.getKey());
            }

            rangeSlicesQuery.setRowCount(rcount);
            QueryResult<OrderedRows<String, String, String>> result = rangeSlicesQuery.execute();
            OrderedRows<String, String, String> orderedRows =
```

```
result.get();

            lastRow = orderedRows.peekLast();
            rowValues = new HashMap();
            int rowf = 0;
            for (Row<String, String, String> row : orderedRows) {
            Map columnValues = new HashMap();
                for(int i = 0;i < ColumnNames.length;i++)
                {
                    String cv = (String)
row.getColumnSlice().getColumnByName(ColumnNames[i]).getValue();
                    columnValues.put(ColumnNames[i], cv);
                }
                rowValues.put(rowf, columnValues);
                rowf++;
            }
        }catch(Exception ex)
        {
            ex.printStackTrace();
        }
       return rowValues;
    }
}
```

D、HectorBaseApi.java

```
import static me.prettyprint.hector.api.factory.HFactory.createColumn;
import static
me.prettyprint.hector.api.factory.HFactory.createColumnQuery;
import static
me.prettyprint.hector.api.factory.HFactory.createMultigetSliceQuery;
import static
me.prettyprint.hector.api.factory.HFactory.createMutator;
import java.util.HashMap;
import java.util.Map;
import me.prettyprint.cassandra.serializers.StringSerializer;
import me.prettyprint.cassandra.service.CassandraHostConfigurator;
import me.prettyprint.hector.api.Cluster;
import me.prettyprint.hector.api.Keyspace;
import me.prettyprint.hector.api.Serializer;
import me.prettyprint.hector.api.beans.HColumn;
import me.prettyprint.hector.api.beans.Rows;
```

```
import me.prettyprint.hector.api.exceptions.HectorException;
import me.prettyprint.hector.api.mutation.Mutator;
import me.prettyprint.hector.api.query.ColumnQuery;
import me.prettyprint.hector.api.query.MultigetSliceQuery;
import me.prettyprint.hector.api.query.QueryResult;

/**
 * Hector对Cassandra操作的部分API
 * @author Hector Dev Group
 *
 */
public class HectorBaseApi {

    public String KEYSPACE = ""; // 相当于数据库
    public String CF_NAME = ""; // 相当于表
    public String ROWKEY = "";
    public String COLUMN_NAME = ""; // 一个列的名称
    public String COLUMN_VALUE = ""; // 一个列的值
    public StringSerializer serializer = StringSerializer.get(); //序列化字符串
    public String hosts = ""; //群集主机列表
    public String clusterName= ""; //群集名称

    public CassandraHostConfigurator cc;//群集主机设置
    public Cluster c;//群集
    public Keyspace keyspace;//键空间变量

    /**
     * Insert a new value keyed by key
     *
     * @param key
     *          Key for the value
     * @param value
     *          the String value to insert
     */
    protected <K> void insert(final K key, final String value,
            Serializer<K> keySerializer) {
        createMutator(keyspace, keySerializer).insert(key, CF_NAME,
                createColumn(COLUMN_NAME, value, serializer,
serializer));
    }

```

```
    /**
     * Get a string value.
     *
     * @return The string value; null if no value exists for the given
key.
     */
    protected <K> String get(final K key, Serializer<K> keySerializer)
            throws HectorException {
        ColumnQuery<K, String, String> q = createColumnQuery(keyspace,
                keySerializer, serializer, serializer);
        QueryResult<HColumn<String, String>> r = q.setKey(key)

    .setName(COLUMN_NAME).setColumnFamily(CF_NAME).execute();
        HColumn<String, String> c = r.get();
        return c == null ? null : c.getValue();
    }

    /**
     * Get multiple values
     *
     * @param keys
     * @return
     */
    protected <K> Map<K, String> getMulti(Serializer<K> keySerializer,
K... keys) {
        MultigetSliceQuery<K, String, String> q =
createMultigetSliceQuery(
                keyspace, keySerializer, serializer, serializer);
        q.setColumnFamily(CF_NAME);
        q.setKeys(keys);
        q.setColumnNames(COLUMN_NAME);

        QueryResult<Rows<K, String, String>> r = q.execute();
        Rows<K, String, String> rows = r.get();
        Map<K, String> ret = new HashMap<K, String>(keys.length);
        for (K k : keys) {
            HColumn<String, String> c = rows.getByKey(k).getColumnSlice()
                    .getColumnByName(COLUMN_NAME);
            if (c != null && c.getValue() != null) {
                ret.put(k, c.getValue());
            }
        }
```

```
        return ret;
    }

    /**
     * Insert multiple values
     */
    protected <K> void insertMulti(Map<K, String>
keyValues,Serializer<K> keySerializer) {
        Mutator<K> m = createMutator(keyspace, keySerializer);
        for (Map.Entry<K, String> keyValue : keyValues.entrySet()) {
            m.addInsertion(
                    keyValue.getKey(),
                    CF_NAME,
                    createColumn(COLUMN_NAME, keyValue.getValue(),
                            keyspace.createClock(), serializer,
serializer));
        }
        m.execute();
    }

    /**
     * Delete multiple values
     */
    protected <K> void delete(Serializer<K> keySerializer, K... keys) {
        Mutator<K> m = createMutator(keyspace, keySerializer);
        for (K key : keys) {
            m.addDeletion(key, CF_NAME, COLUMN_NAME, serializer);
        }
        m.execute();
    }
}
```

E、TestCassandraClient.java

```
import java.util.Map;

public class TestCassandraClient {

    public static String hosts = "127.0.0.1:9160";
    public static String clusterName = "iCluster";
    public static String keyspaceName = "CassandraKS";
    public static String columnFamily = "Products";
```

```
    @SuppressWarnings("rawtypes")
    public static void main(String[] args) {

        //参数装载
        CassandraParas ps = new CassandraParas();
        ps.setHosts(hosts); //注解数据
        ps.setClusterName(clusterName);//群集名称
        ps.setKeyspaceName(keyspaceName);//键空间
        ps.setColumnFamily(columnFamily);//列族

        CassandraClient cc = new CassandraClient(ps);

//      String[] columnNames =
{"Id_Product","Product_Name","Id_Factory",
//              "Factory_Name","Product_Type","Product_Place",
//              "Product_Price","Product_Count","Product_Income",
//              "Product_Batch"};
//
        //String[] columnNames =
{"Id_Product","Product_Name","Product_Type","Product_Batch"};
        //查询开始
//      //查询产品表返回产品类型是"Computer",进货日期大于"2011/10/15"的数据
//      String[] exColumnNames =
{"rflag","Product_Type","Product_Income"};
//      String[] expressions =    {"=","=",">"};
//      String[] exColumnvalues = {"1","Computer","2011/10/15"};
//      Map rowValues  = cc.getMultiColumnByIndex(columnNames,
exColumnNames, exColumnvalues, expressions);
//      for(int i = rowValues.size()-1;i >= 0;i--)
//      {
//          Map columnValues = (Map) rowValues.get(i);
//          for(int j = 0;j < columnValues.size() ;j++)
//              System.out.print("[" + columnNames[j] + " = " +
columnValues.get(columnNames[j]) + "],");
//          System.out.println("");
//      }
        //查询 结束

        String[] columnNames =
{"Id_Product","Product_Name","Product_Type","Product_Batch"};
```

```
        System.out.println("--向下翻页，每页2行，每一页的开始行都是上一页的结束行--");
        Map rowValues = cc.getMultiPage(2, columnNames,1);
        for(int i = 0;i < rowValues.size();i++)
        {
            Map columnValues = (Map) rowValues.get(i);
            for(int j = 0;j < columnValues.size() ;j++)
                System.out.print("[" + columnNames[j] + " = " + columnValues.get(columnNames[j]) + "],");
            System.out.println("");
        }

        System.out.println("--向下翻页，每页2行，每一页的开始行都是上一页的结束行--");
        rowValues = cc.getMultiPage(2, columnNames,1);
        for(int i = 0;i < rowValues.size();i++)
        {
            Map columnValues = (Map) rowValues.get(i);
            for(int j = 0;j < columnValues.size() ;j++)
                System.out.print("[" + columnNames[j] + " = " + columnValues.get(columnNames[j]) + "],");
            System.out.println("");
        }

        System.out.println("--向上翻页，每页2行，每一页的结束行都是上一页的开始行--");
        rowValues = cc.getMultiPage(2, columnNames,0);
        for(int i = 0;i < rowValues.size();i++)
        {
            Map columnValues = (Map) rowValues.get(i);
            for(int j = 0;j < columnValues.size() ;j++)
                System.out.print("[" + columnNames[j] + " = " + columnValues.get(columnNames[j]) + "],");
            System.out.println("");
        }

        //测试单个字段插入和读取

//      String key = "kexi002";
//      String columnName = "Last_Name";
//      String columnValue = "克喜";
//      boolean inf = cc.insertOneColumn(key,columnName,columnValue);
```

```
//      if(inf)
//      {
//          System.out.println(cc.getOneColumn(key,columnName));
//      }
//
//      //测试多个字段插入和读取
//      String key = "3";//后面说怎么做自动递增的key
//      String[] columnNames =
{"Id_Product","Product_Name","Id_Factory",
//              "Factory_Name","Product_Type","Product_Place",
//              "Product_Price","Product_Count","Product_Income",
//              "Product_Batch"};
//      String[] columnValues = {"3","乐PAD A107","1","联想","移动通讯",
//              "天津","890.00","300","2011/10/12","T-C"};
//
//      boolean inf = cc.insertMultiColumn(key, columnNames,
columnValues);
//      if(inf)
//      {
//          Map map = new HashMap();
//          map = cc.getMultiColumn(key, columnNames);
//          for(int i = 0;i < map.size() ;i++)
//              System.out.println(columnNames[i] + " : " +
map.get(columnNames[i]));
//      }
//
//      //删除一个字段
//      inf = cc.deleteOneColumn(key, columnName);
//      if(inf)
//      {
//          System.out.println(cc.getOneColumn(key,columnName));
//      }
//
//      String[] delcolumnNames = {"First_Name","Last_Name","Address"};
//      inf = cc.deleteMultiColumn(key, delcolumnNames);
//      if(inf)
//      {
//          System.out.println("OK");
//      }

    }
```

```
}
```